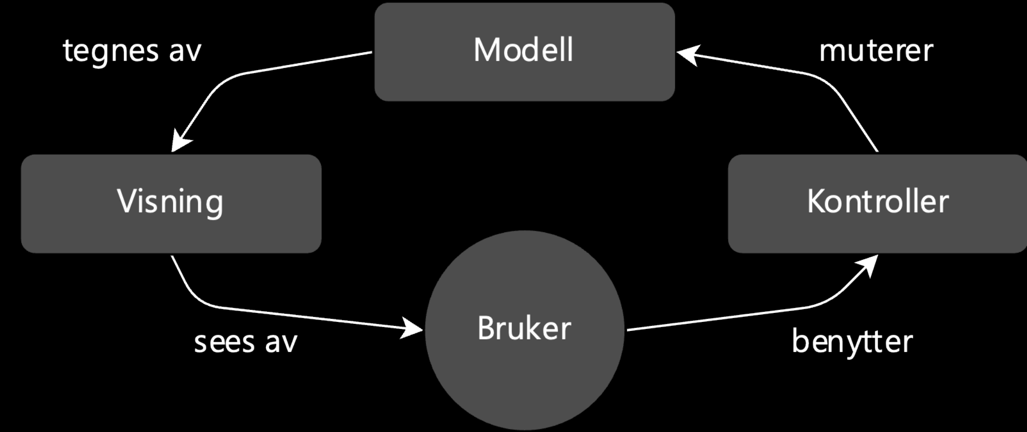
Observer

INF101 forelesning 24. mars 2023

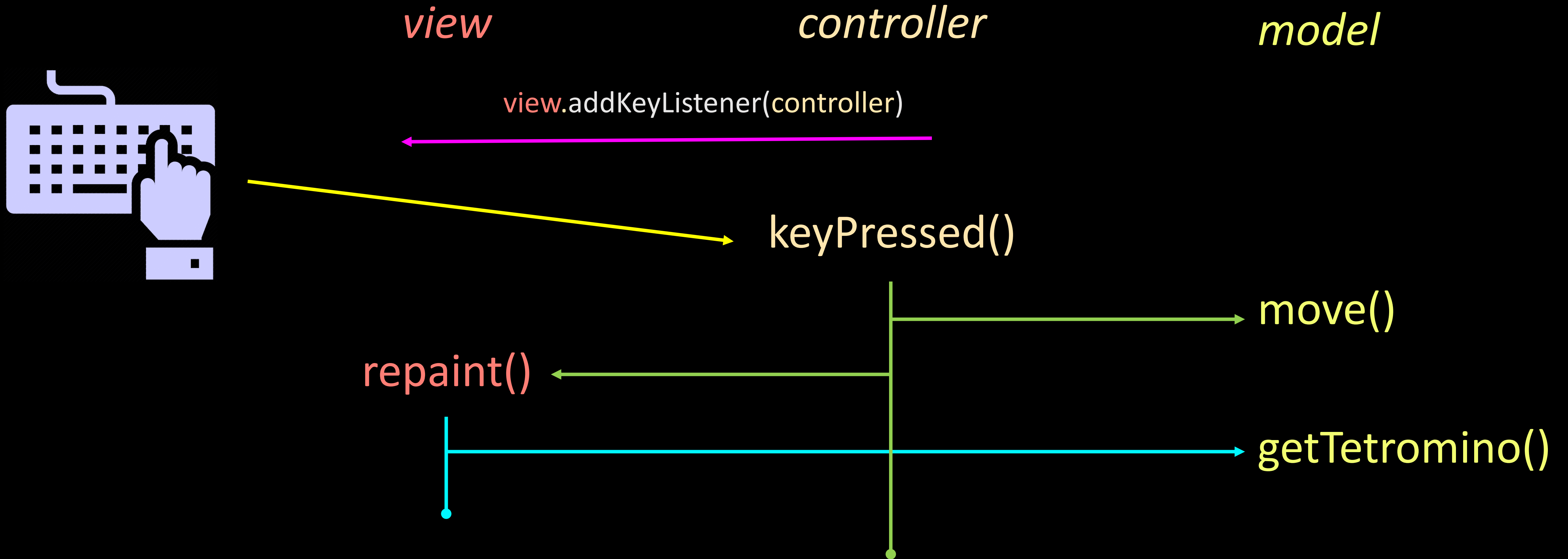
Torstein Strømme

Tetris: model-view-controller

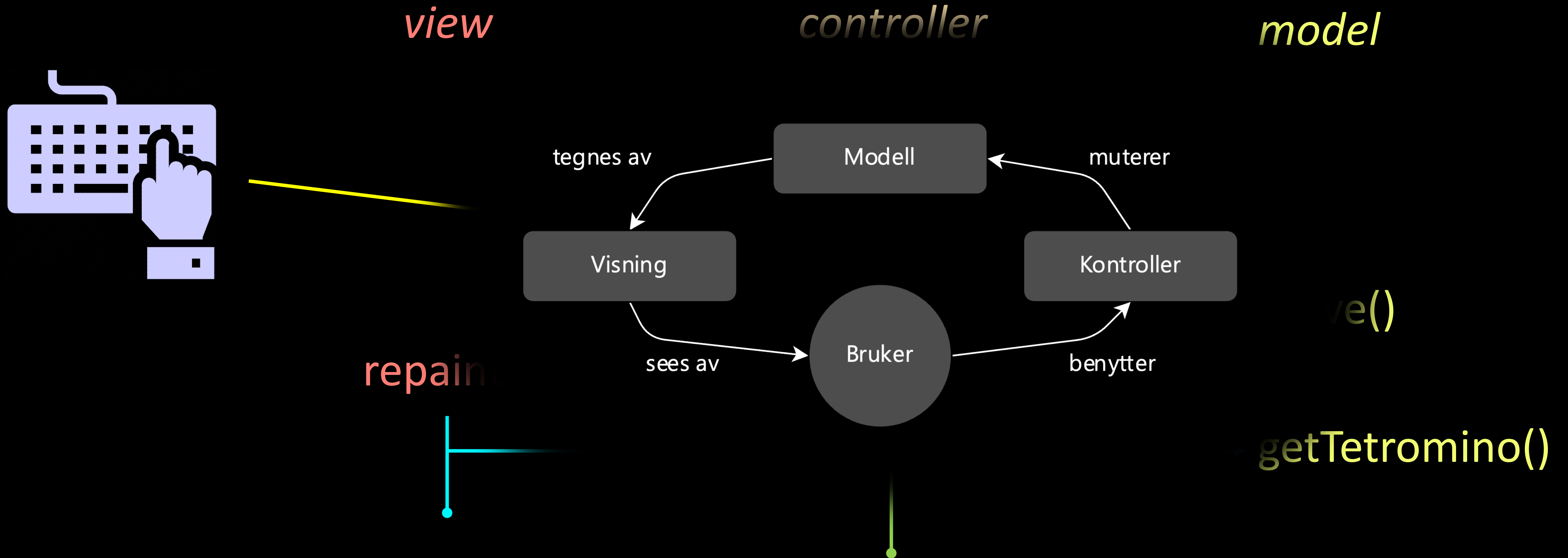
- Modellen
  - Lagrer informasjon
  - Regler for modifisering av informasjon
- Kontroller
  - Bestemmer hvordan input skal påvirke modellen
- Visning
  - Viser modellen



Tetris: model-view-controller



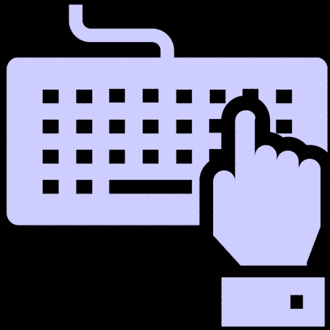
Tetris: model-view-controller

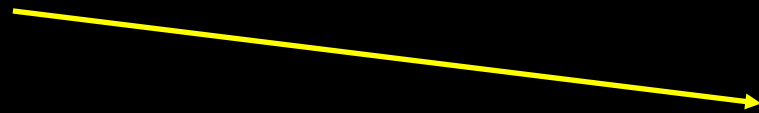


Er det egentlig bra at *kontrollen* kaller repaint?

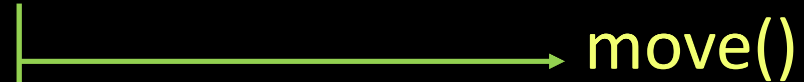
Tetris: model-view-controller

view *controller* *model*





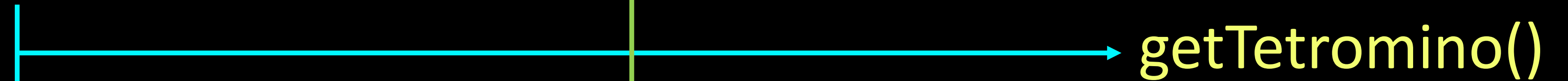
keyPressed()



move()

?

repaint()



getTetromino()

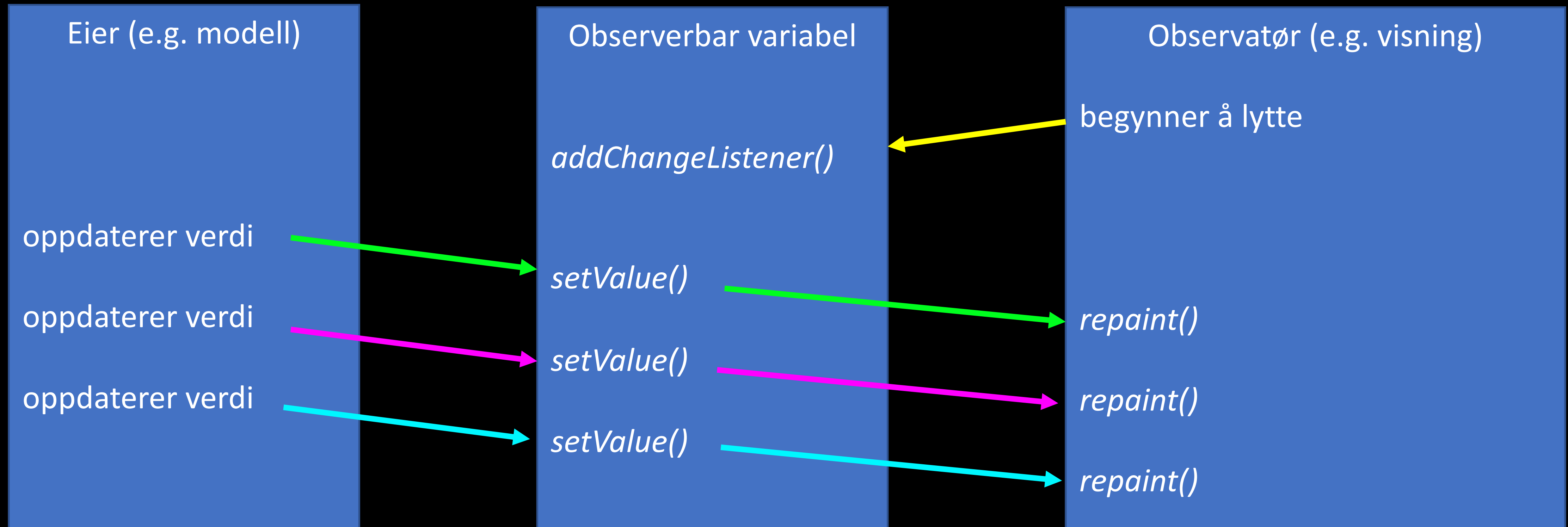
Bør *modellen* kalle repaint?

- Modellen *bør* kalle repaint fordi:
  - Modellen vet best om den har endret seg
  - Kontrollen bør slippe å tenke på visningen
  - Uansett hvilken kontroller som endret modellen, blir view oppdatert

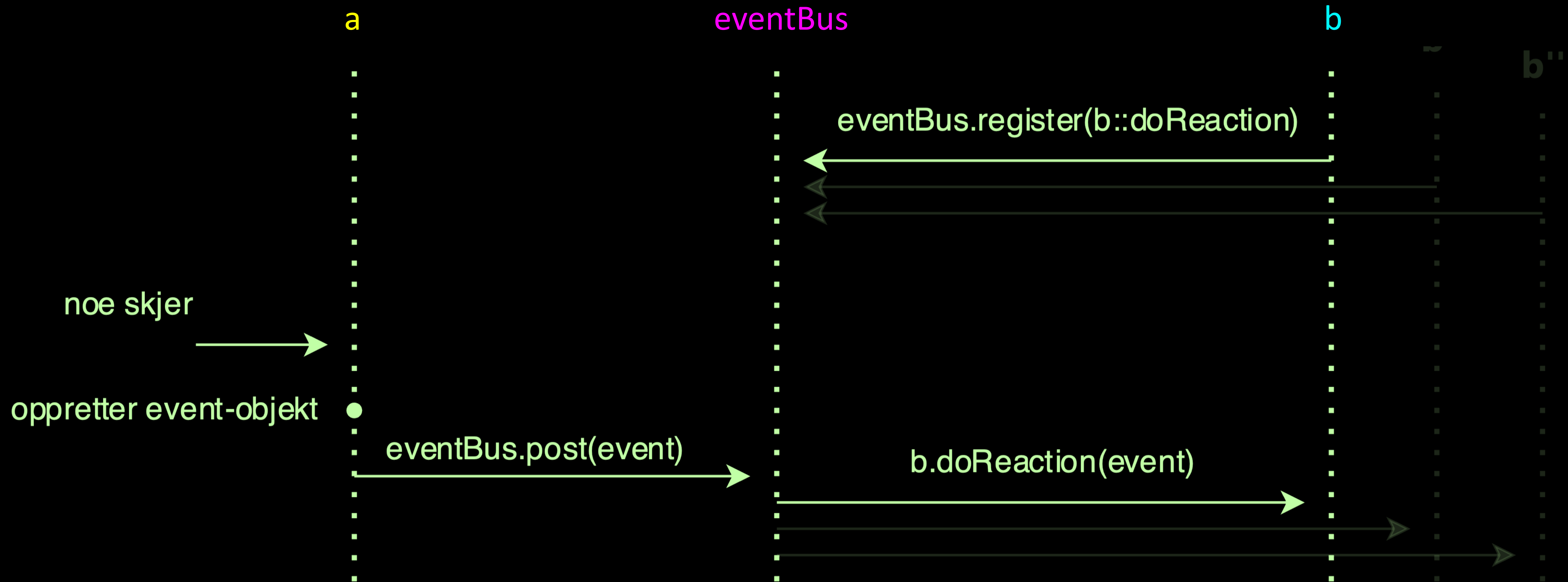
- Modellen bør *ikke* kalle repaint
  - Modellen bør slippe å tenke på visningen
  - Modellen vet ikke hvilke deler av modellen som faktisk vises

Løsning: observerbare variabler

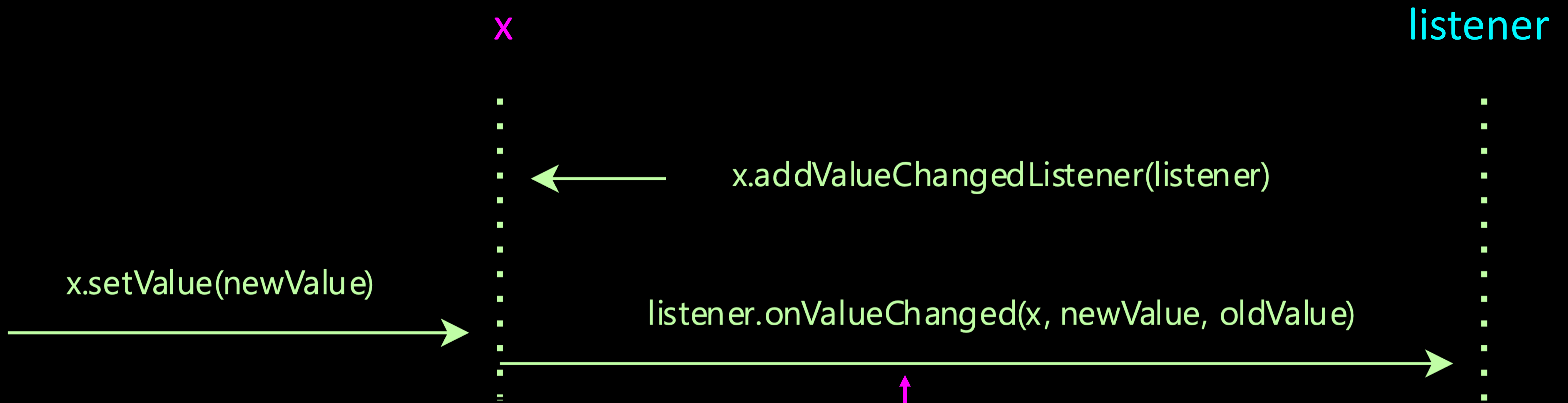
- En observerbar variabel er en variabel man kan «abonnere på endringer ved»



Tilbakeblikk: Eventbuss

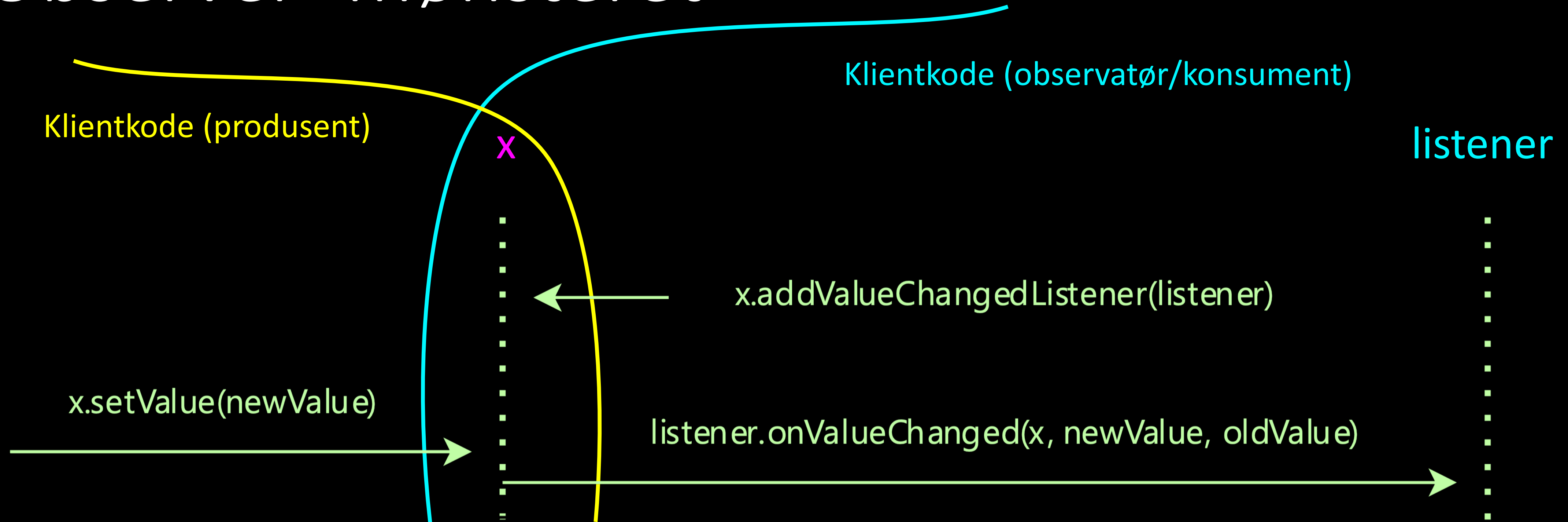


Observer -mønsteret



En *observerbar* variabel genererer hendelser når den endrer verdi

Observer -mønsteret



En *observerbar* variabel genererer hendelser når den endrer verdi

Observerbart heltall

- I utgangspunktet bare en «boks» for en verdi

```java
public class ObservableInt {

  private int value;

  public ObservableInt(int initialValue) {
    this.value = initialValue;
  }



  public int getValue() {
    return this.value;
  }


  public void setValue(int newValue) {
    this.value = newValue;
  }




}
```

Observerbart heltall

- I utgangspunktet bare en «boks» for en verdi
- Har også en liste over lyttere.

```
public class ObservableInt {
  private final List<ValueChangedListener> listeners = new ArrayList<>();
  private int value;

  public ObservableInt(int initialValue) {
    this.value = initialValue;
  }



  public int getValue() {
    return this.value;
  }


  public void setValue(int newValue) {
    this.value = newValue;
  }




}
```

Observerbart heltall

- I utgangspunktet bare en «boks» for en verdi
- Har også en liste over lyttere.
- Lyttere kan registreres

```java
public class ObservableInt {
  private final List<ValueChangedListener> listeners = new ArrayList<>();
  private int value;

  public ObservableInt(int initialValue) {
    this.value = initialValue;
  }

  public void addValueChangedListener(ValueChangedListener listener) {
    this.listeners.add(listener);
  }

  public int getValue() {
    return this.value;
  }


  public void setValue(int newValue) {
    this.value = newValue;
  }







}
```

Observerbart heltall

- I utgangspunktet bare en «boks» for en verdi
- Har også en liste over lyttere.
- Lyttere kan registreres
- Lytterne blir kalt hver gang verdien endres

```
public class ObservableInt {
  private final List<ValueChangedListener> listeners = new ArrayList<>();
  private int value;

  public ObservableInt(int initialValue) {
    this.value = initialValue;
  }

  public void addValueChangedListener(ValueChangedListener listener) {
    this.listeners.add(listener);
  }

  public int getValue() {
    return this.value;
  }


  public void setValue(int newValue) {
    this.value = newValue;
    this.notifyListeners();
  }

  private void notifyListeners() {
    for (ValueChangedListener listener : this.listeners) {
      listener.onValueChanged();
    }
  }
}
```

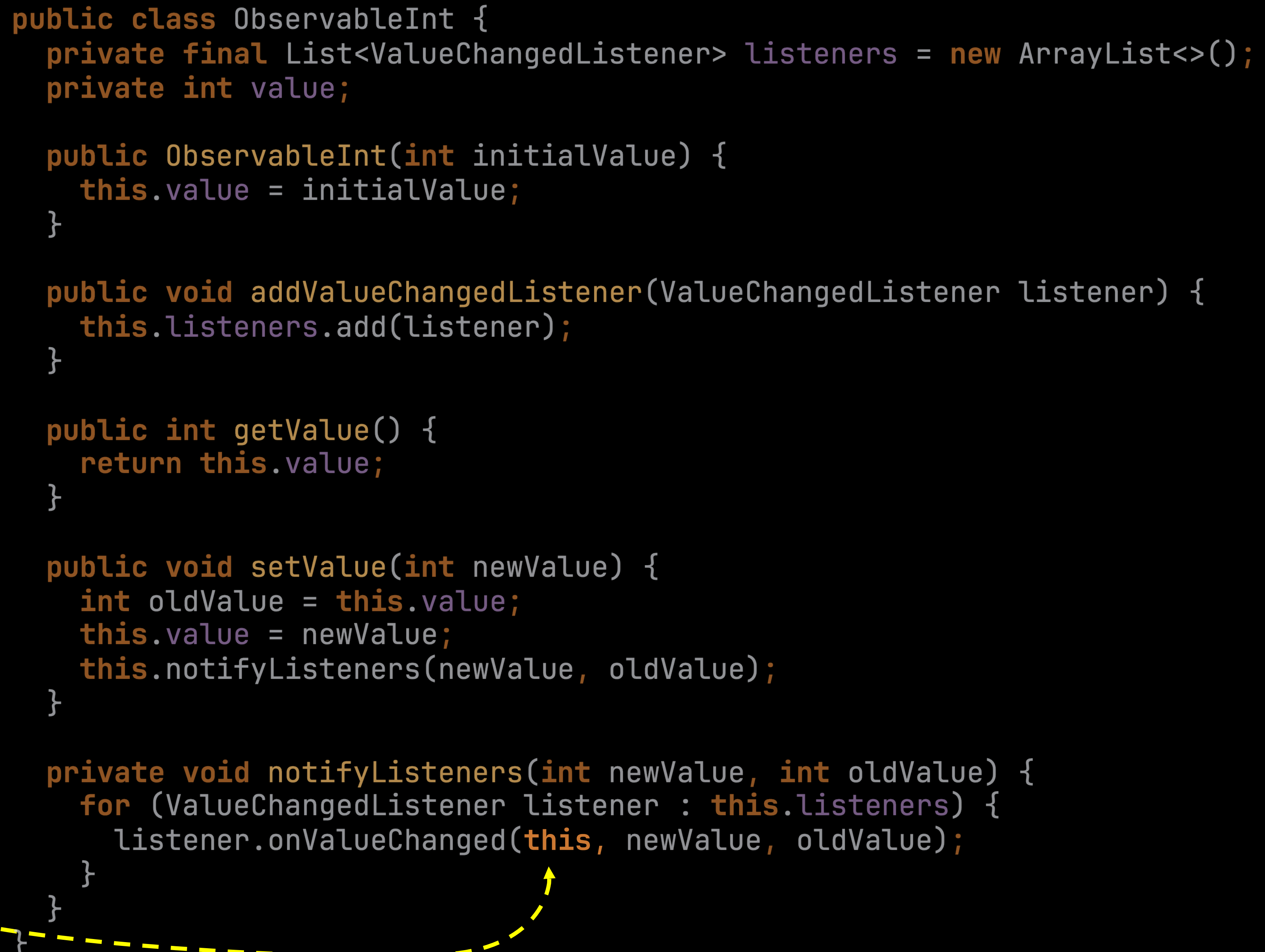
Observerbart heltall

- I utgangspunktet bare en «boks» for en verdi
- Har også en liste over lyttere.
- Lyttere kan registreres
- Lytterne blir kalt hver gang verdien endres
- Lyttere får informasjon om ny og gammel verdi

```java
public class ObservableInt {
  private final List<ValueChangedListener> listeners = new ArrayList<>();
  private int value;

  public ObservableInt(int initialValue) {
    this.value = initialValue;
  }

  public void addValueChangedListener(ValueChangedListener listener) {
    this.listeners.add(listener);
  }

  public int getValue() {
    return this.value;
  }

  public void setValue(int newValue) {
    int oldValue = this.value;
    this.value = newValue;
    this.notifyListeners(newValue, oldValue);
  }

  private void notifyListeners(int newValue, int oldValue) {
    for (ValueChangedListener listener : this.listeners) {
      listener.onValueChanged(newValue, oldValue);
    }
  }
}
```

Observerbart heltall

- I utgangspunktet bare en «boks» for en verdi
- Har også en liste over lyttere.
- Lyttere kan registreres
- Lytterne blir kalt hver gang verdien endres
- Lyttere får informasjon om ny og gammel verdi (og hvilket objekt det er)



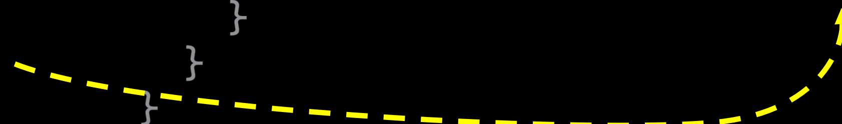
```java
public class ObservableInt {
  private final List<ValueChangedListener> listeners = new ArrayList<>();
  private int value;

  public ObservableInt(int initialValue) {
    this.value = initialValue;
  }

  public void addValueChangedListener(ValueChangedListener listener) {
    this.listeners.add(listener);
  }

  public int getValue() {
    return this.value;
  }

  public void setValue(int newValue) {
    int oldValue = this.value;
    this.value = newValue;
    this.notifyListeners(newValue, oldValue);
  }

  private void notifyListeners(int newValue, int oldValue) {
    for (ValueChangedListener listener : this.listeners) {
      listener.onValueChanged(this, newValue, oldValue);
    }
  }
}
```

Observerbart heltall

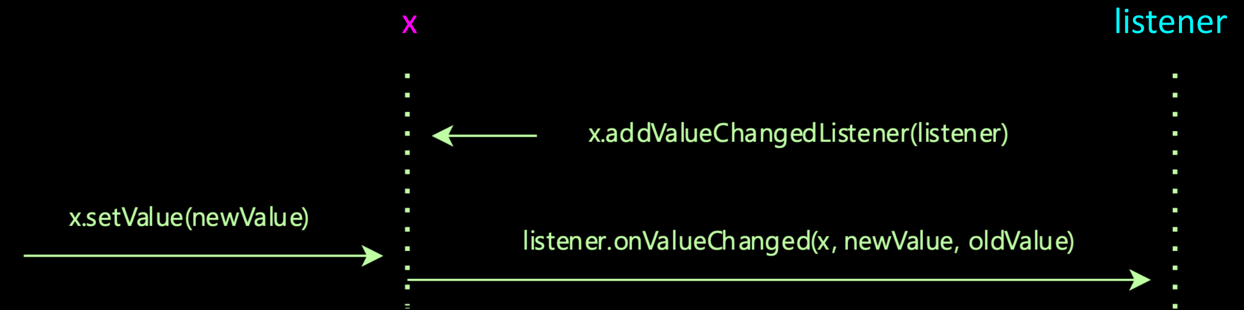
- I utgangspunktet bare en «boks» for en verdi
- Har også en liste over lyttere.
- Lyttere kan regis
- Lytterne blir kalt gang verdien endres
- Lyttere får informasjon om ny og gammel verdi (og hvilket objekt det er)

```
public class ObservableInt {
  private final List<ValueChangedListener> listeners = new ArrayList<>();
  private int value;

  public ObservableInt(int initialValue) {
    this.value = initialValue;
  }

  public void addValueChangedListener(ValueChangedListener listener) {
    this.listeners.add(listener);
  }

  public int getValue() {
    return this.value;
  }

  private void notifyListeners(int newValue, int oldValue) {
    for (ValueChangedListener listener : this.listeners) {
      listener.onValueChanged(this, newValue, oldValue);
    }
  }
}
```

```
public interface ValueChangedListener {
    void onValueChanged(ObservableInt source, int newValue, int oldValue);
}
```

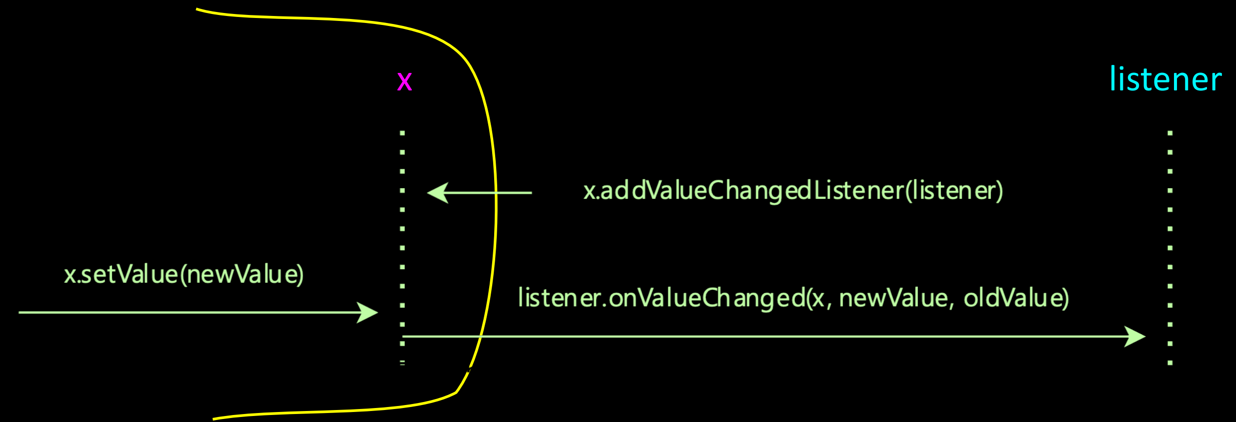


Observerbart heltall




```
public class Main {
  public static void main(String[] args) {
    ClientProducer producer = new ClientProducer();
    ClientConsumer consumer = new ClientConsumer(producer);

    producer.changeStuff();
    producer.changeStuff();
    producer.changeStuff();
  }
}
```

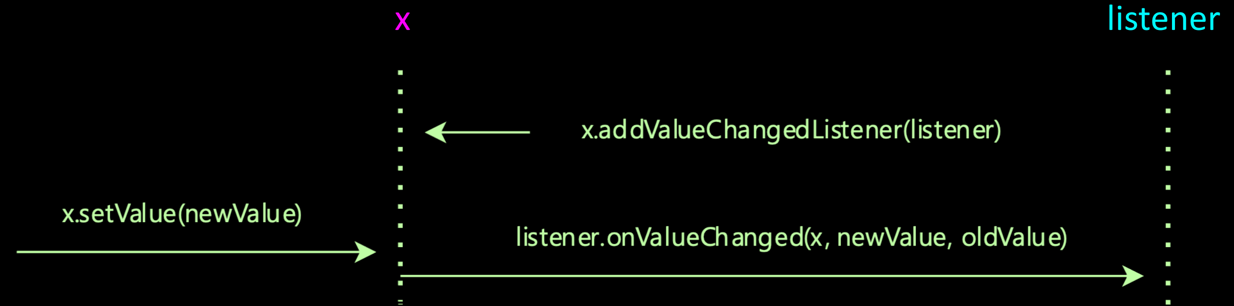
Observerbart heltall




```java
public class Main {

}
```

```java
public class ClientProducer {

  private final ObservableInt variable = new ObservableInt(0);

  public void changeStuff() {
    int currentValue = this.variable.getValue();
    int nextValue = currentValue + 1;
    this.variable.setValue(nextValue);
  }

  public ObservableInt getVariable() {
    return this.variable;
  }
}
```

Observerbart heltall

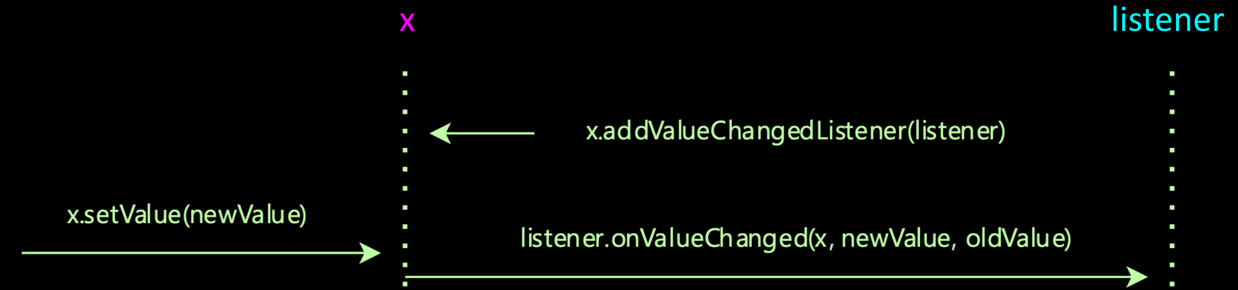



```java
public class Main {
}
```

```java
public class ClientProducer {
}
```

```java
public class ObservableInt {
  private final List<ValueChangedListener> listeners = new ArrayList<>();
  private int value;
  // ...
  public void setValue(int newValue) {
    int oldValue = this.value;
    this.value = newValue;
    this.notifyListeners(newValue, oldValue);
  }

  private void notifyListeners(int newValue, int oldValue) {
    for (ValueChangedListener listener : this.listeners) {
      listener.onValueChanged(this, newValue, oldValue);
    }
  }
}
```

Observerbart heltall



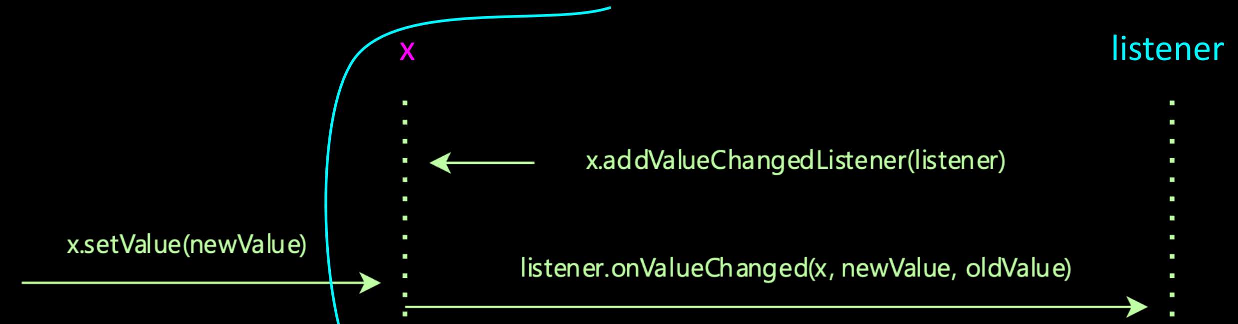

```java
public interface ValueChangedListener {
  void onValueChanged(ObservableInt source, int newValue, int oldValue);
}
```

```java
public class Main {
}
```

```java
public class ClientProduc
}
```

```java
public class ObservableInt {
  private final List<ValueChangedListener> listeners = new ArrayList<>();
  private int value;
  // ...
  public void setValue(int newValue) {
    int oldValue = this.value;
    this.value = newValue;
    this.notifyListeners(newValue, oldValue);
  }

  private void notifyListeners(int newValue, int oldValue) {
    for (ValueChangedListener listener : this.listeners) {
      listener.onValueChanged(this, newValue, oldValue);
    }
  }
}
```

Observerbart heltall




```
public interface ValueChangedListener {
  void onValueChanged(ObservableInt source, int newValue, int oldValue);
}
```

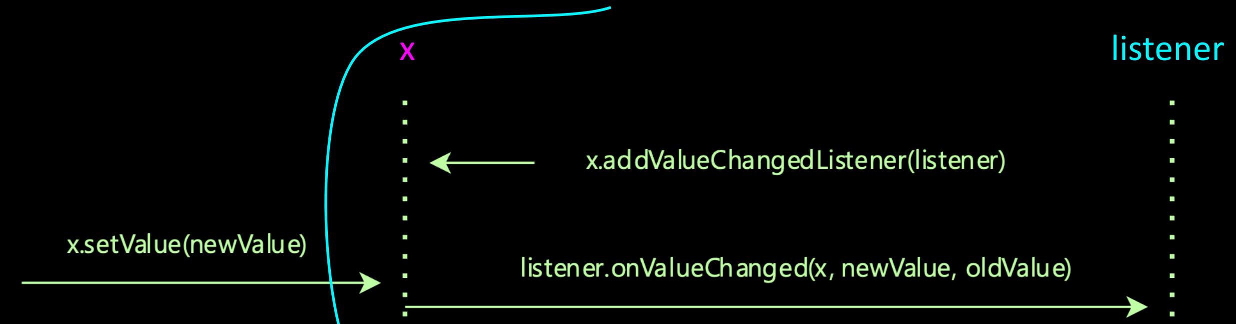
```
public class Main {
}
```

```
public class ClientProduc
}
```

```
public class ObservableInt {
      listener.onValueChanged(this, newValue, oldValue);
    }
  }
}
```

```
public class ClientConsumer {

  public ClientConsumer(ClientProducer producer) {
    ObservableInt variable = producer.getVariable();
    variable.addValueChangedListener(this::listener);
  }

  private void listener(ObservableInt source, int newValue, int oldValue) {
    System.out.println("Value changed from " + oldValue + " to " + newValue);
  }
}
```

Observerbart Foo




```java
public interface ValueChangedListener<E> {
  void onValueChanged(ObservableValue<E> source, E newValue, E oldValue);
}
```

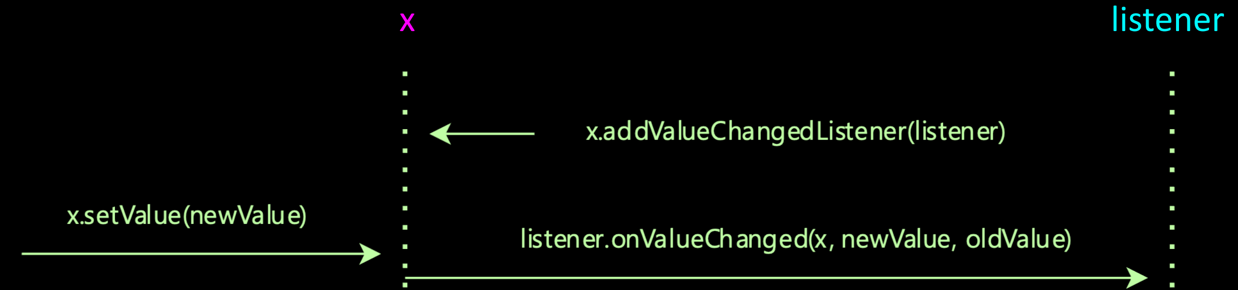
```java
public class Main {
}
```

```java
public class ClientProduc
}
```

```java
public class ObservableValue<E> {
      listener.onValueChanged(this, newValue, oldValue);
    }
  }
}
```

```java
public class ClientConsumer {

  public ClientConsumer(ClientProducer producer) {
    ObservableValue<Foo> variable = producer.getVariable();
    variable.addValueChangedListener(this::listener);
  }

  private void listener(ObservableValue<Foo> source, Foo newValue, Foo oldValue) {
    System.out.println("Value changed from " + oldValue + " to " + newValue);
  }
}
```

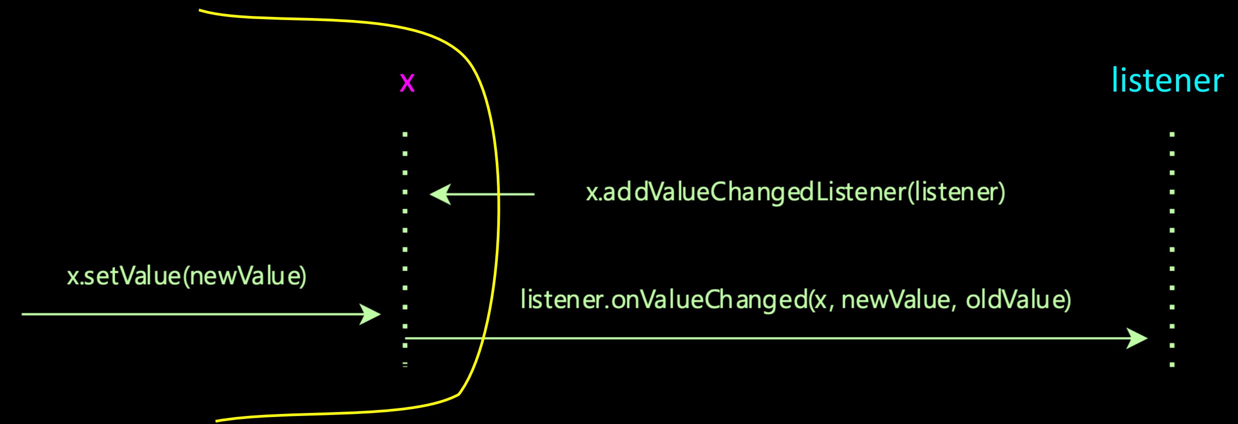
Observerbart Foo




```
public interface ValueChangedListener<E> {
  void onValueChanged(ObservableValue<E> source, E newValue, E oldValue);
}
```

```
public class Main {
  public class ClientProduc
    public class ObservableValue<E> {
      private final List<ValueChangedListener<E>> listeners = new ArrayList<>();
      private E value;
      // ...
      public void setValue(E newValue) {
        E oldValue = this.value;
        this.value = newValue;
        this.notifyListeners(newValue, oldValue);
      }

      private void notifyListeners(E newValue, E oldValue) {
        for (ValueChangedListener listener : this.listeners) {
          listener.onValueChanged(this, newValue, oldValue);
        }
      }
    }
  }
}
```

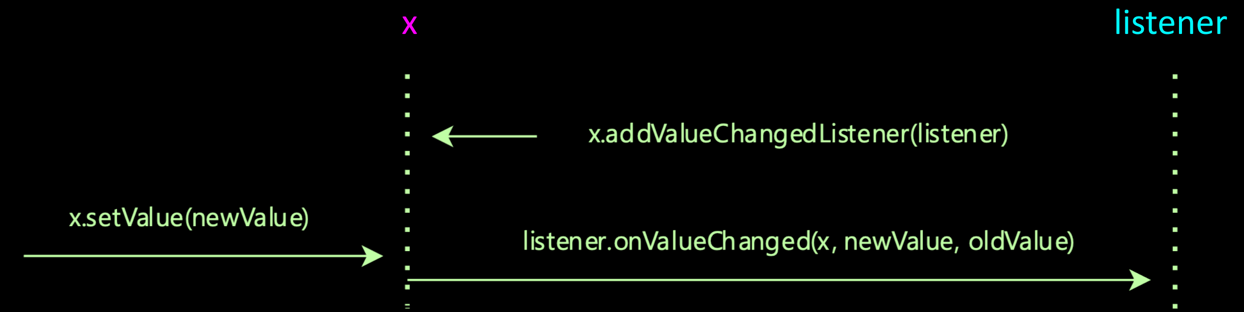
Observerbart Foo




```java
public class Main {

}
```

```java
public class ClientProducer {

  private final ObservableValue<Foo> variable = new ObservableValue<>(new Foo());

  public void changeStuff() {
    Foo currentValue = this.variable.getValue();
    Foo nextValue = currentValue.getModifiedCopy();
    this.variable.setValue(nextValue);
  }

  public ObservableValue<Foo> getVariable() {
    return this.variable;
  }
}
```

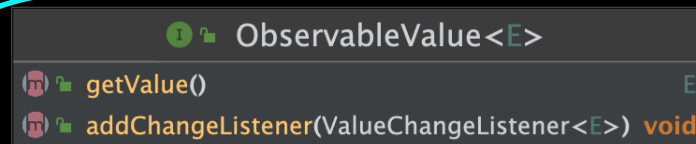
Observerbart Foo



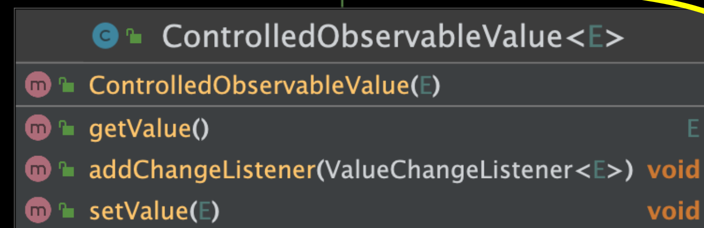

```java
public class Main {
  public static void main(String[] args) {
    ClientProducer producer = new ClientProducer();
    ClientConsumer consumer = new ClientConsumer(producer);

    producer.changeStuff();
    producer.changeStuff();
    producer.changeStuff();
  }
}
```

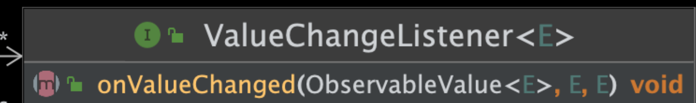
Observer med restriktivt grensesnitt

- Bør observatøren ha mulighet til å endre variabelen?
  - Kommer an på
  - Noen ganger er det ikke ønskelig. For eksempel: i model-view-controller skal ikke visningen endre modellen
  - Løsning: del opp ObservableValue i to: grensesnitt + implementasjon



Observatører/konsumenter bruker denne typen



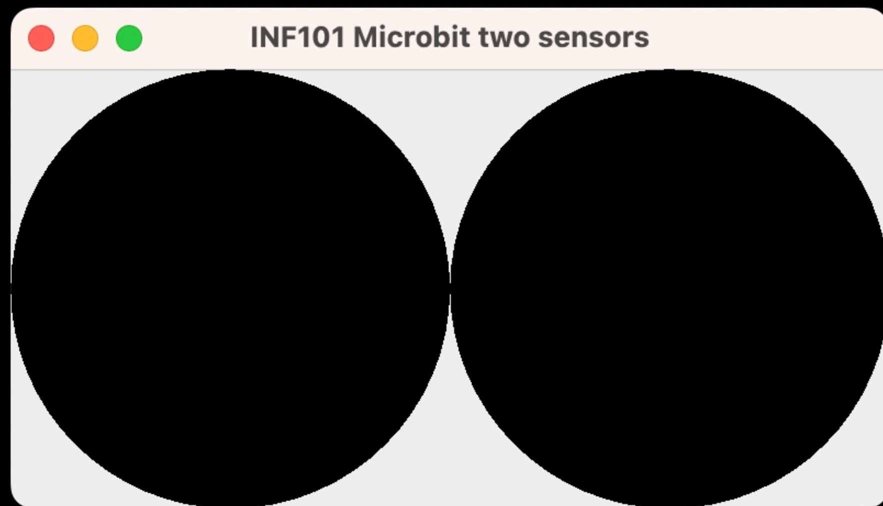


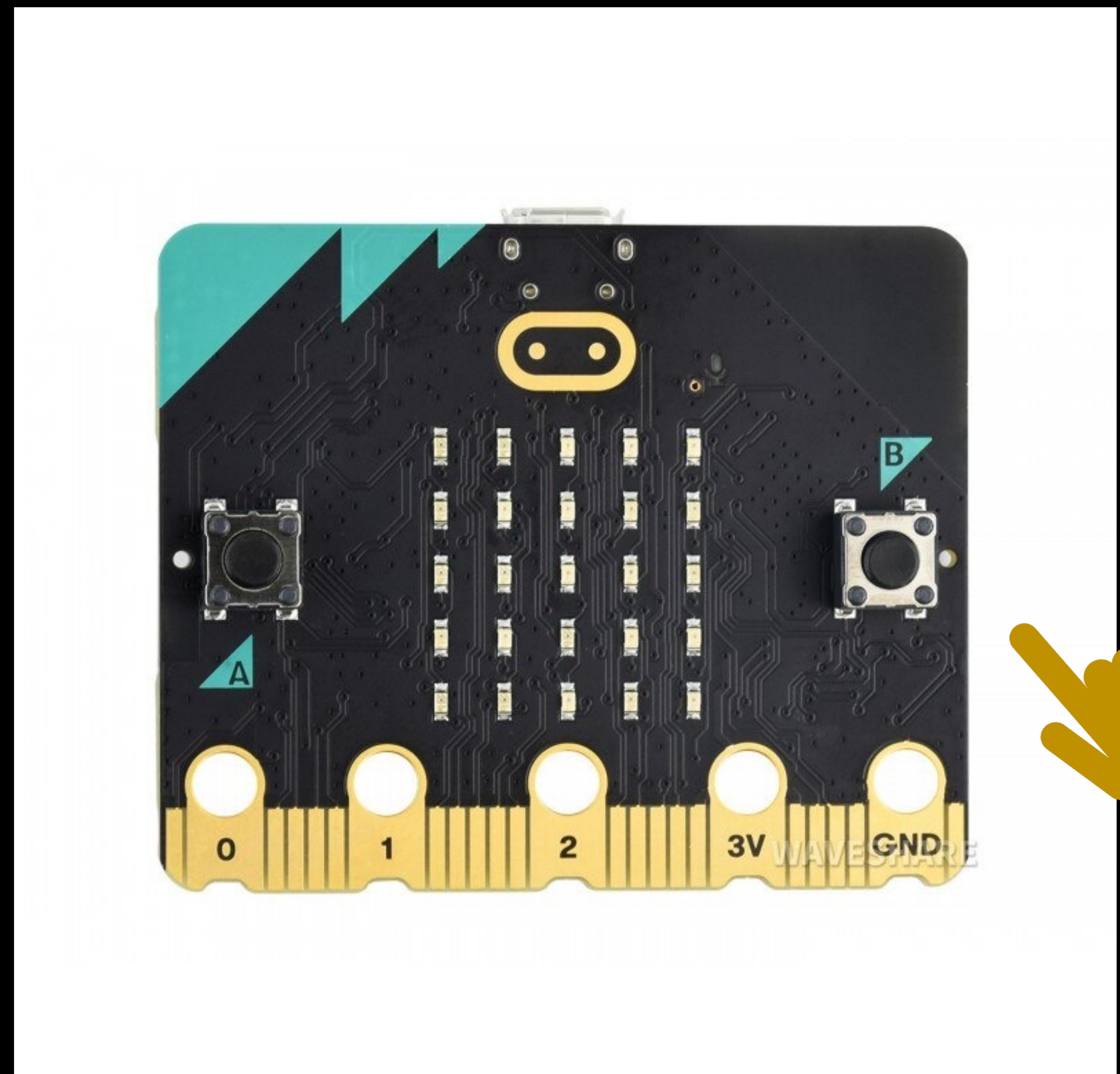
Produsenter bruker denne typen

Observer med restriktivt grensesnitt

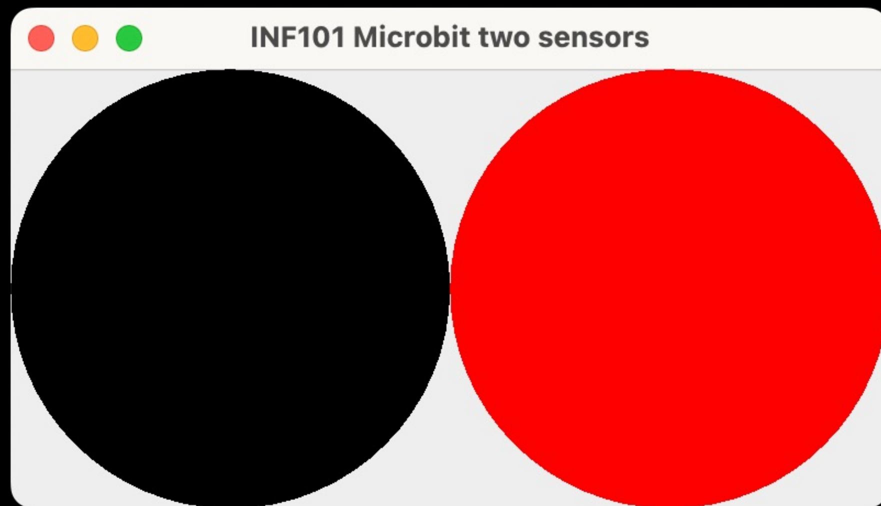
```java
public class ClientProducer {

  private final ControlledObservableValue<Foo> variable = new ControlledObservableValue<>(new Foo());

  public void changeStuff() {
    Foo currentValue = this.variable.getValue();
    Foo nextValue = currentValue.getModifiedCopy();
    this.variable.setValue(nextValue);
  }

  public ObservableValue<Foo> getVariable() {
    return this.variable;
  }
}
```

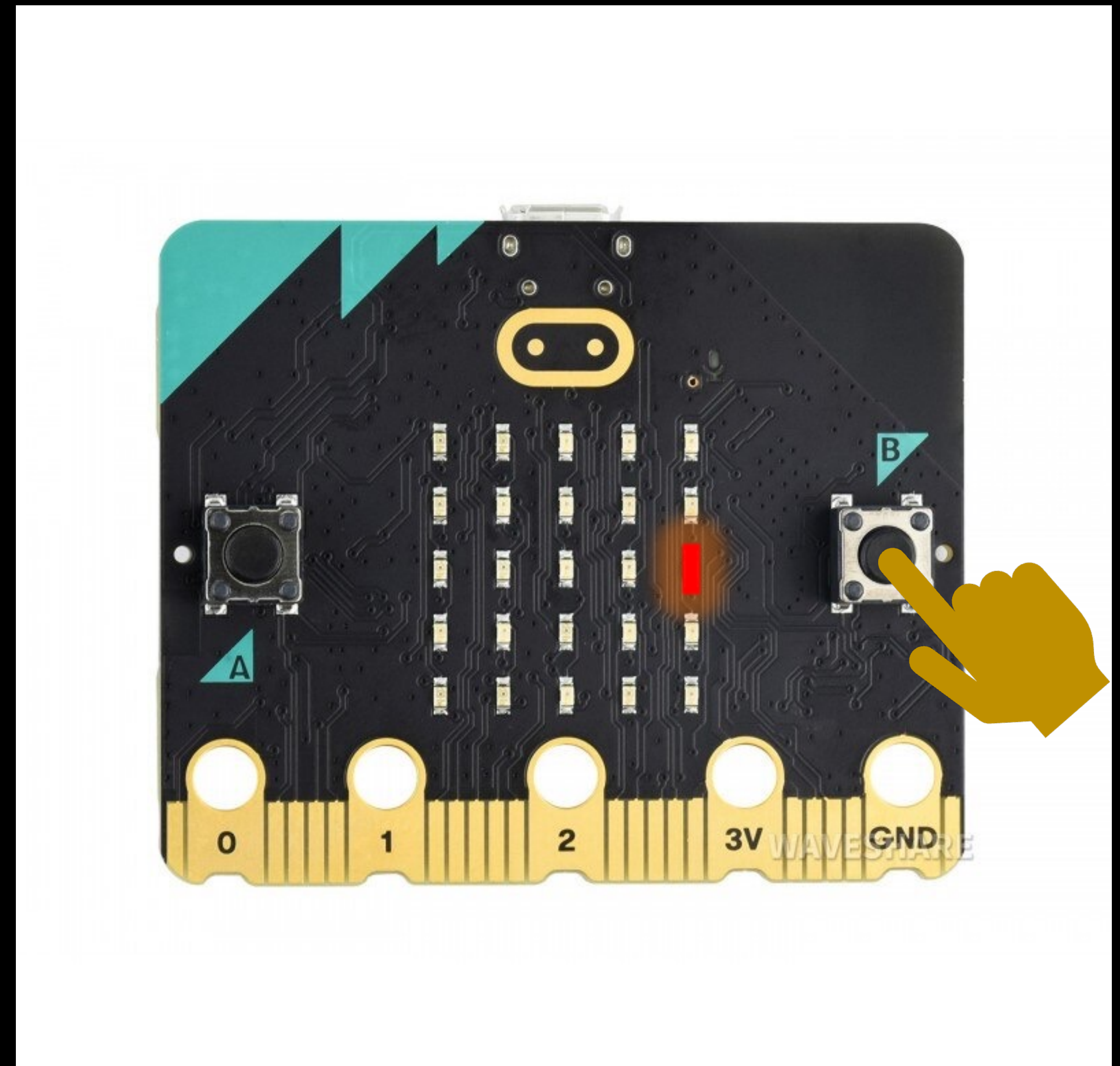
Eksempel: en knapp



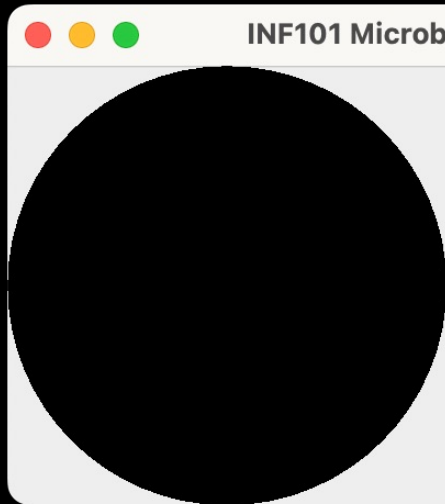


Eksempel: en knapp



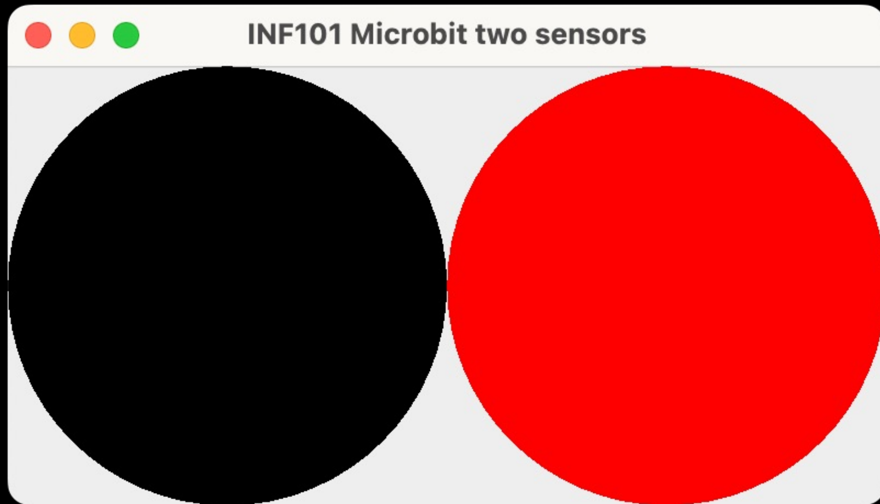


Eksempel: en knapp



- Opprett en modell (ModelSensors)
  - Boolean feltvariabel som er observerbar og kontrollerbar
- Opprett en visning (ViewSensor)
  - Motta Boolean som skal observeres i konstruktør
  - Registrer som observatør
  - Tegn en runding: rød hvis boolean er true og svart ellers
- Opprett en kontroller (ControllerSensor)
  - Bruk SerialComListener for å lytte etter signaler; omgjør signalene fra char til kall på modellen som endrer verdiene

Eksempel: en knapp



- La modellen ha to feltvariabler
- Opprett en «hoved-visning» som benytter to ViewSensor-paneler